Flexi Medical Dispatch

Streamlining Emergency Response

by Aroh Ebenezer

Project Links

Project Links

- **Live App - https://flexi-med-app-0be2aab2ec6b.herokuapp.com/**
-
- **ERD on Dbdiagram - https://dbdiagram.io/d/FlexiMed-67dd449975d75cc844f63b30**
- **Code Repository - https://github.com/Benny-Nwaro/flexi-med.git**
- **Full Code Documentation - https://flexi-med-app-0be2aab2ec6b.herokuapp.com/docs/index.html**
- **Swagger API Docs - https://flexi-med-app-0be2aab2ec6b.herokuapp.com/swagger-ui/index.html**

The Challenge of Efficient Dispatch

- Rapid response is critical in medical emergencies; every second counts.
- Traditional dispatch systems often rely on manual tracking, increasing the risk of delays and human error.
- Uncoordinated communication between dispatchers, drivers, and hospitals limits operational efficiency.
- The app addresses real-world issues such as resource misallocation, lack of tracking, and fragmented data handling.

Project Overview

- A real-time ambulance dispatch and tracking system that streamlines emergency response using location tracking, WebSockets, and user-friendly interfaces. Built with **Spring Boot (Backend)** and **React + Leaflet (Frontend)**.

Tech Stack

- **Backend**: Spring Boot, WebSockets (STOMP/SockJS), JPA, PostgreSQL, Google Oauth, Google SMTP for notifications, Twilio SMS service, Together service AI for first aid treatment on arrival etc
- **Frontend**: React, Leaflet.js, Axios, WebSockets
- **Database**: PostgreSQL
- **Others**: Swagger for API docs, dbdiagram.io for ERD, GitHub for version control

Backend Technologies Used

- Spring Boot – Rapid backend development framework with dependency injection.
- Spring Data JPA – ORM for database interaction with cleaner, declarative syntax.
- PostgreSQL – Relational database with strong support for spatial data types.
- REST API – Communication standard for frontend/backend integration.
- JWT – Secure token-based authentication for protecting routes.
- Java – Primary language for logic-heavy and concurrent applications.
- WebSocket – Live communication for notification, tracking and status updates.

Core Functionalities

- Real-time Ambulance Tracking: Uses GPS data pushed via WebSocket for dispatcher live monitoring.
- Automated Dispatch: Backend logic evaluates ambulance proximity and availability before assigning.
- Patient Record Management: Includes personal info, emergency notes, and dispatch history securely stored.
- User Roles & Access: Admin (manage all), Dispatcher (Register ambulances and drivers, assign drivers to ambulance, manage ambulance records, drivers records and service history), Driver (Receive dispatch updates, track location on map, complete request once patient is safely in a hospital, Request ambulance, receive notification on dispatch, track ambulance on map, manage patient record/medical history).

Core Functionalities/2

- Emergency Logging: Timestamped logs of request creation, dispatch and completion.

- Push Notifications: Real-time updates sent to requesting users and drivers via WebSocket, email and sms.

Architectural Overview

- Client Interfaces: Web mobile-responsive UI for dispatchers, drivers and users/patients.
- Spring Boot REST API: Manages business logic, authentication, and database access.
- Database Layer: PostgreSQL with entities like Ambulance, Request, ServiceHistory, PatientRecords and User.
- Location Services: Integrated with GPS to fetch and store location updates.
- WebSocket Integration: Enables live updates on ambulance movement and request status.
- Authentication: Secured with JWT and role-based access filtering.

Leveraging Spring Boot for Speed and Reliability

- Robust backend system built on Spring Boot for efficient handling of concurrent requests.
- Microservices-ready architecture with potential for future expansion (e.g., billing, analytics, hospital modules).
- RESTful API design to support frontend applications and third-party systems like GPS services and hospital EHRs.

Data Management

- Entities: User, Ambulance, Request, ServiceHistory, PatientRecord.
- User table supports roles with RBAC (ADMIN, DISPATCHER, DRIVER, USER).
- Ambulance tracks location, status (available, busy), and assigned requests.
- Request table logs incident location, timestamp, and current status.
- ServiceHistory links ambulance and request data for reporting and auditing.
- PatientRecord stores and updates patients medical history.

Data Management

ERD Design

Data Management

Explaining the Relationships

Users Table

- Stores information about all users in the system (patients, admins, drivers).
- **Key Fields**: `user_id`, `name`, `email`, `phone`, `role`.
- **Role-based access** is managed through the `role` field (e.g., admin, user, dispatcher, driver).

Ambulance Table

- Represents ambulances in the system.
- **Each ambulance** is linked to a **driver (user)** via `driver_id`.
- Stores the current status, location, and driver info.

Relationship:

`Ambulance.driver_id` → references `Users.user_id`

(*Each ambulance has at least one driver, who is a registered user*)

Data Management

Explaining the Relationships

Request Table

- Logs ambulance requests made by users (patients).
- Tracks the assigned ambulance, request status, and time-based events.

Relationships:

- `Request.user_id` → references `Users.user_id`
  (*Each request is initiated by a user*)
- `Request.ambulance_id` → references `Ambulance.ambulance_id`
  (*Each request is assigned to one ambulance*)

PatientRecord Table

- Stores medical details related to each ambulance request.
- Tied directly to a **single request**.

Relationship:

- `PatientRecord.request_id` → references `Request.request_id`
  (*Each patient record is linked to one ambulance request*)

Data Management

Explaining the Relationships

ServiceHistory Table

- Captures a log of events throughout a request's lifecycle (e.g., dispatched, arrived).
- Helps in **tracking and auditing** ambulance services.

Relationship:

`ServiceHistory.request_id` → references `Request.request_id`

(*Each entry is tied to a specific request*)

Secure Access

- JWT for stateless, scalable authentication with token expiry.
- Role-Based Access Control (RBAC) ensures users only access allowed endpoints.

Ensuring Reliability

- Unit tests for service layer components.
- Integration tests for API endpoints.
- Load testing to evaluate system performance.
- Explain the test coverage.

Ensuring Reliability

Unit Tests – Testing Individual Pieces

- **Focus**: Service layer components (business logic).
- **Goal**: Ensure each method behaves correctly in isolation.
- **Mocked Dependencies**: Repositories, external services, etc.

- **Example**:
  - Verifying ambulance availability logic.

    Validating request creation under different conditions.

Ensures **core logic** is solid and bug-free before integration.

Ensuring Reliability

Integration Tests – Testing the Flow

- **Focus**: End-to-end API endpoint validation.
- **Goal**: Test how components work together (Controller ↔ Service ↔ DB).
- **Tools**: `@SpringBootTest`, `TestRestTemplate`, or MockMvc.

- **Example**:
  - Creating a request and retrieving it.
  - Testing user registration and authentication flow.

Validates that **data flows correctly** through the application stack.

Next Steps

- Implement real-time traffic updates for route optimization.
- Integrate with hospital systems for seamless patient transfer.
- Develop mobile applications for drivers and dispatchers.
- Add a reporting dashboard.

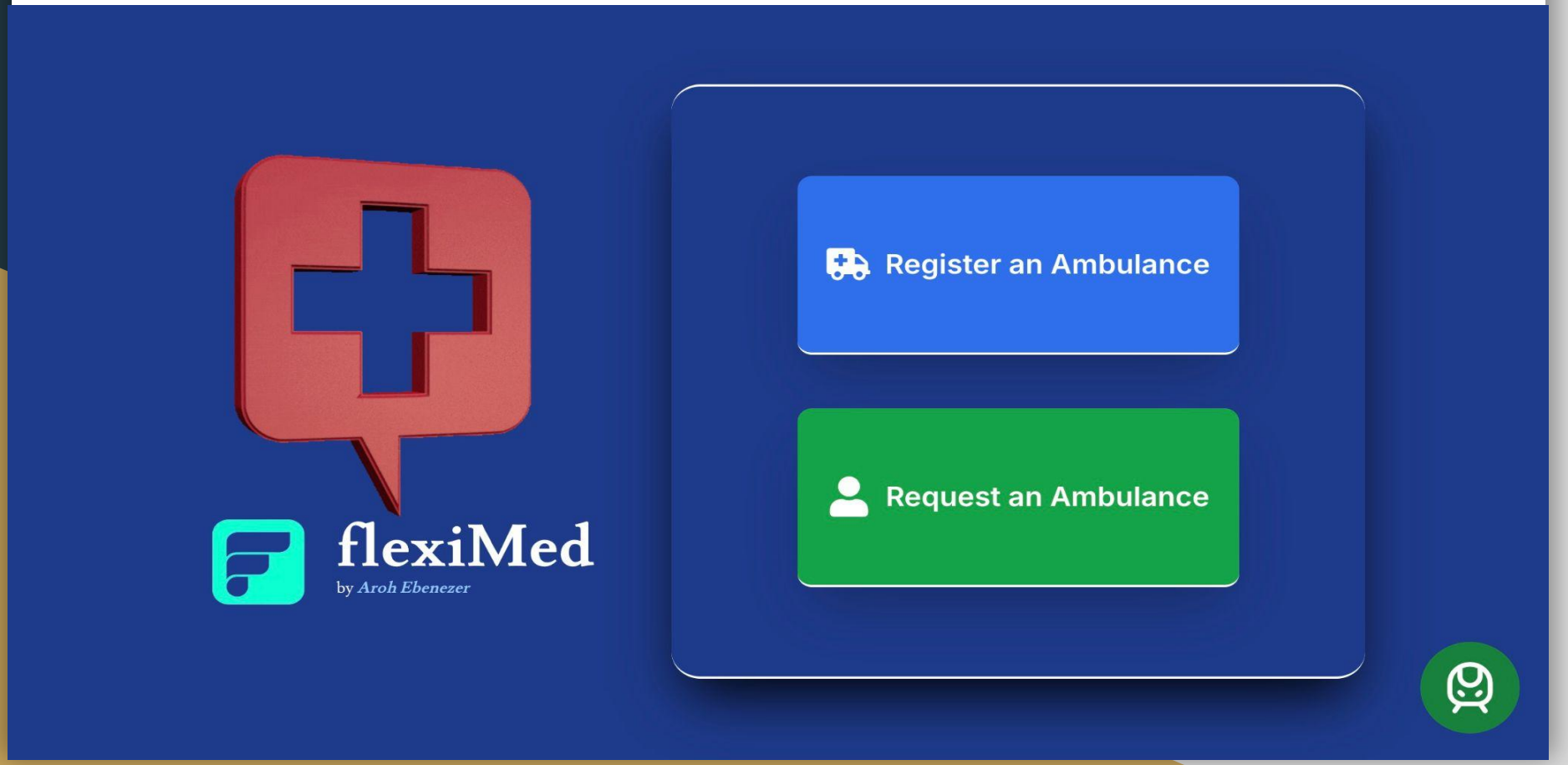
Live Demonstration

Landing page

Live Demonstration

Home page

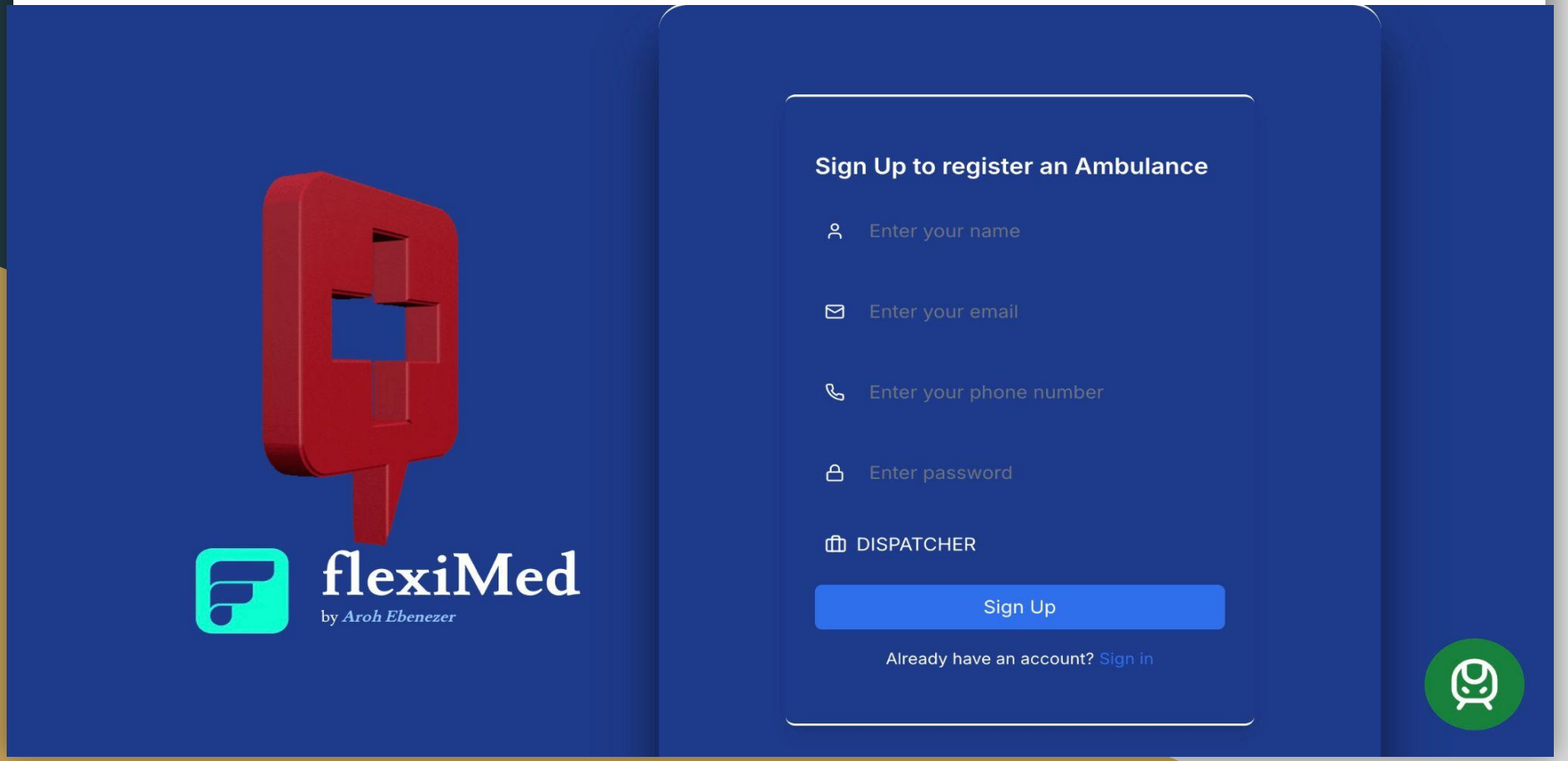
Click a button to register or log in



Live Demonstration

Registration page

Selects a default role(DISPATCHER or USER) depending on th request button clicked



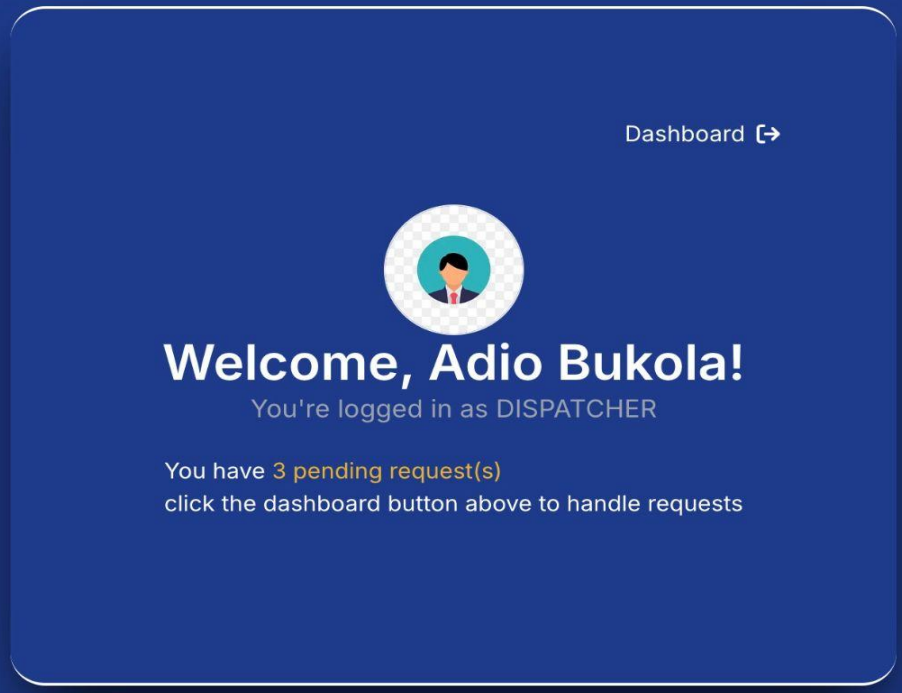
Live Demonstration

User Home page

Dynamically displays users basic details including role and a button to take user to users dashboard

flexiMed

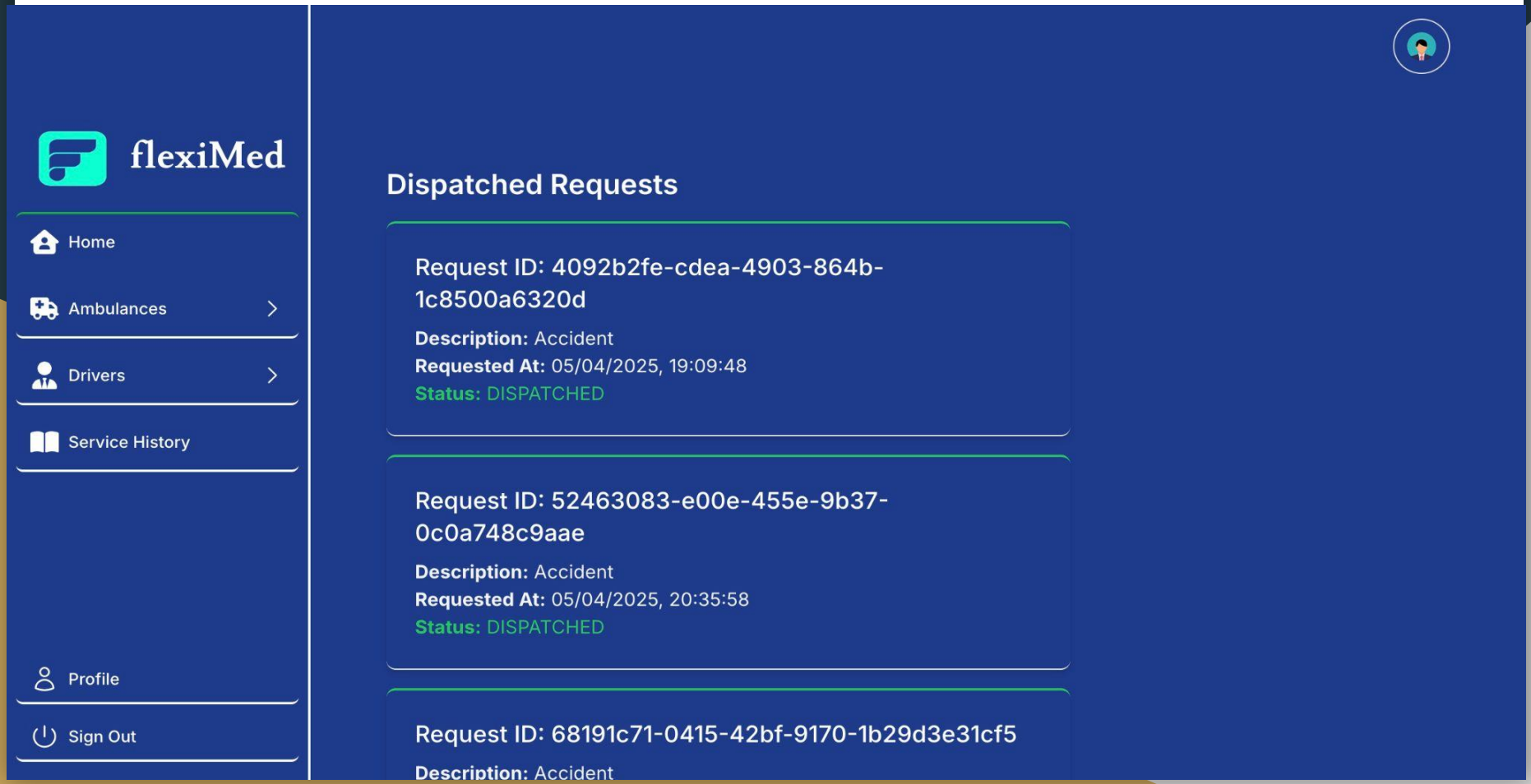
by *Aroh Ebenezer*



Live Demonstration

Users dashboard page

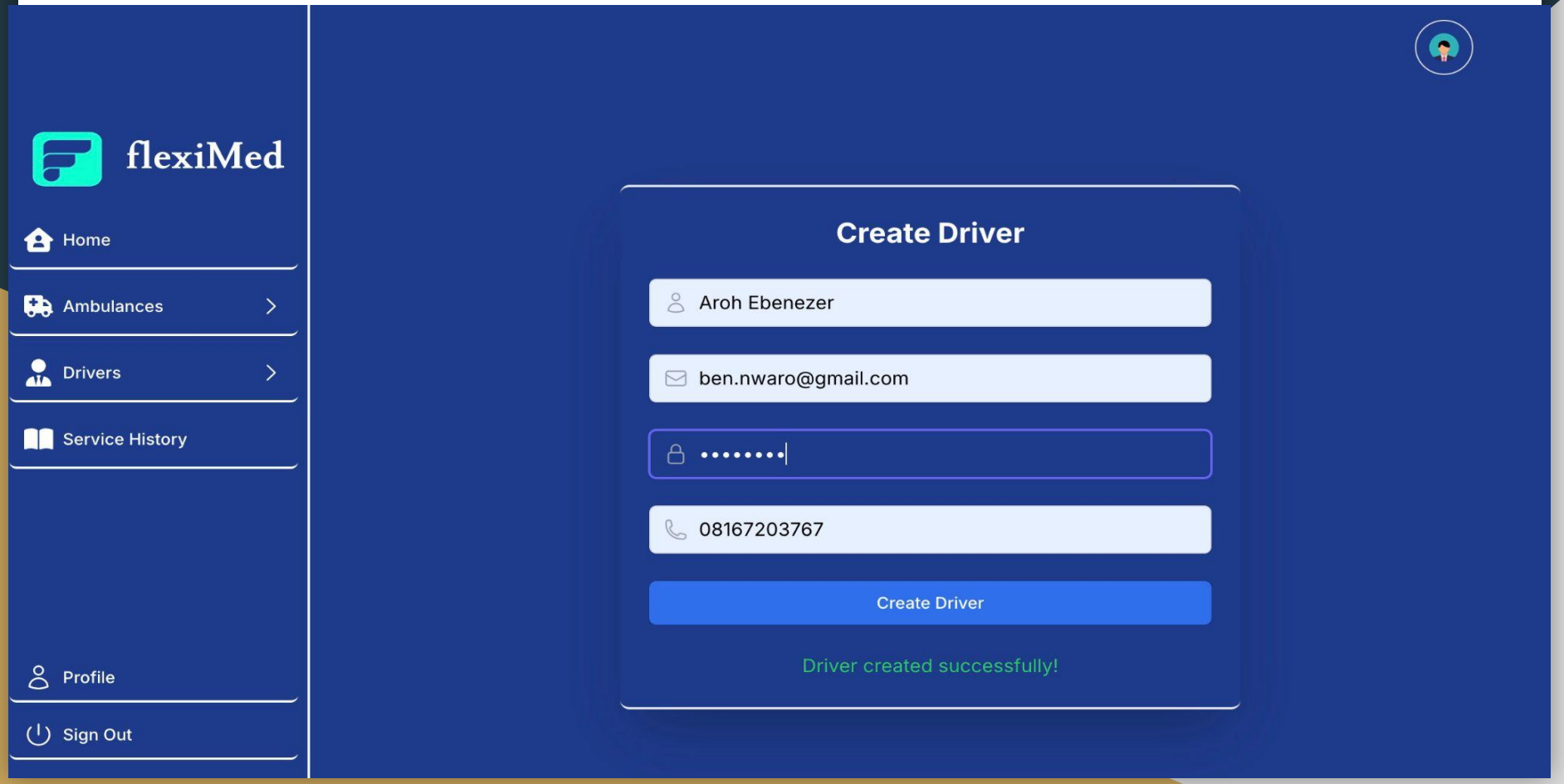
Gives user access to some operations depending on users role



Live Demonstration

Create Driver page

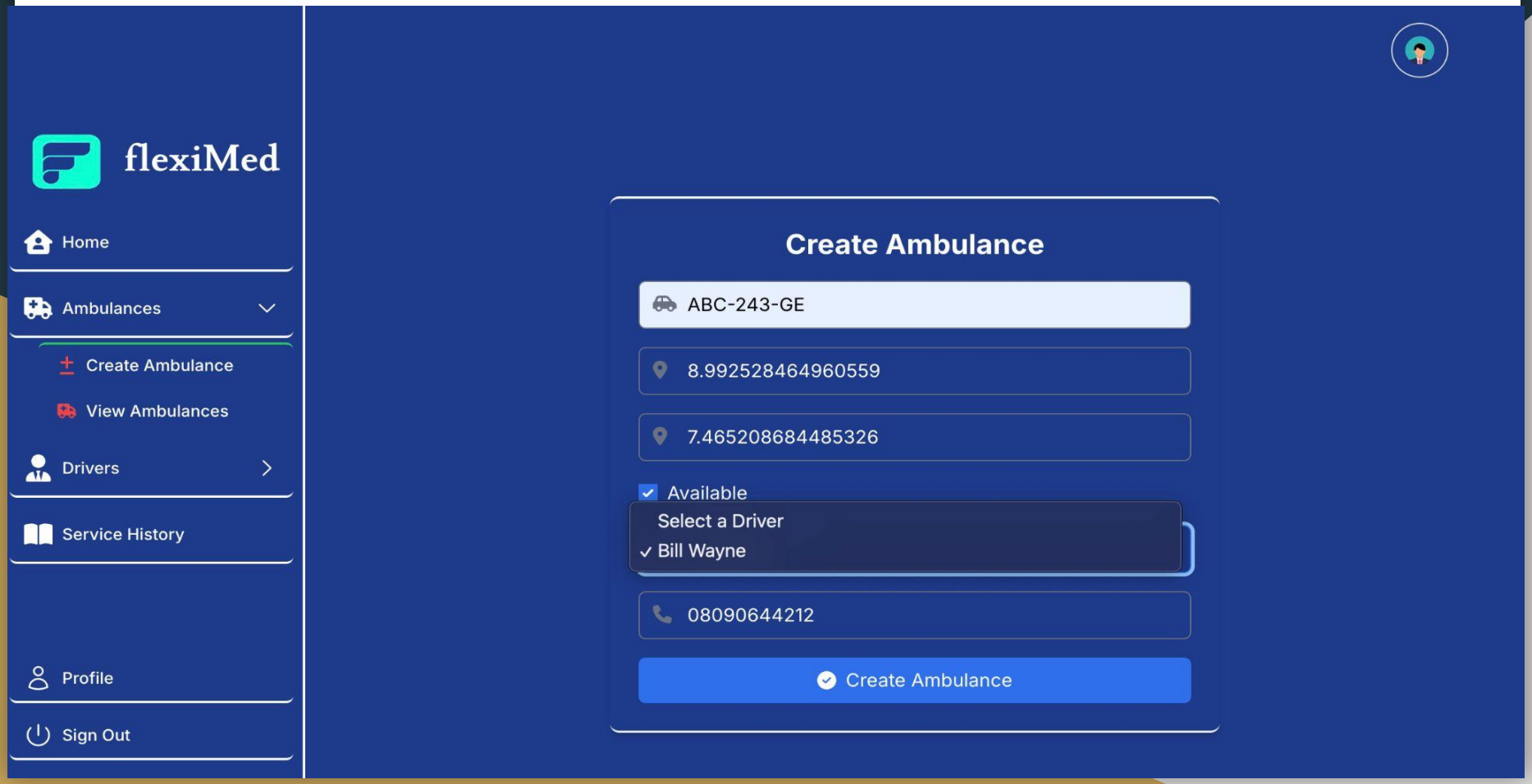
A user with role(Dispatcher) can register a driver and assign the driver to an ambulance



Live Demonstration

Create Ambulance page

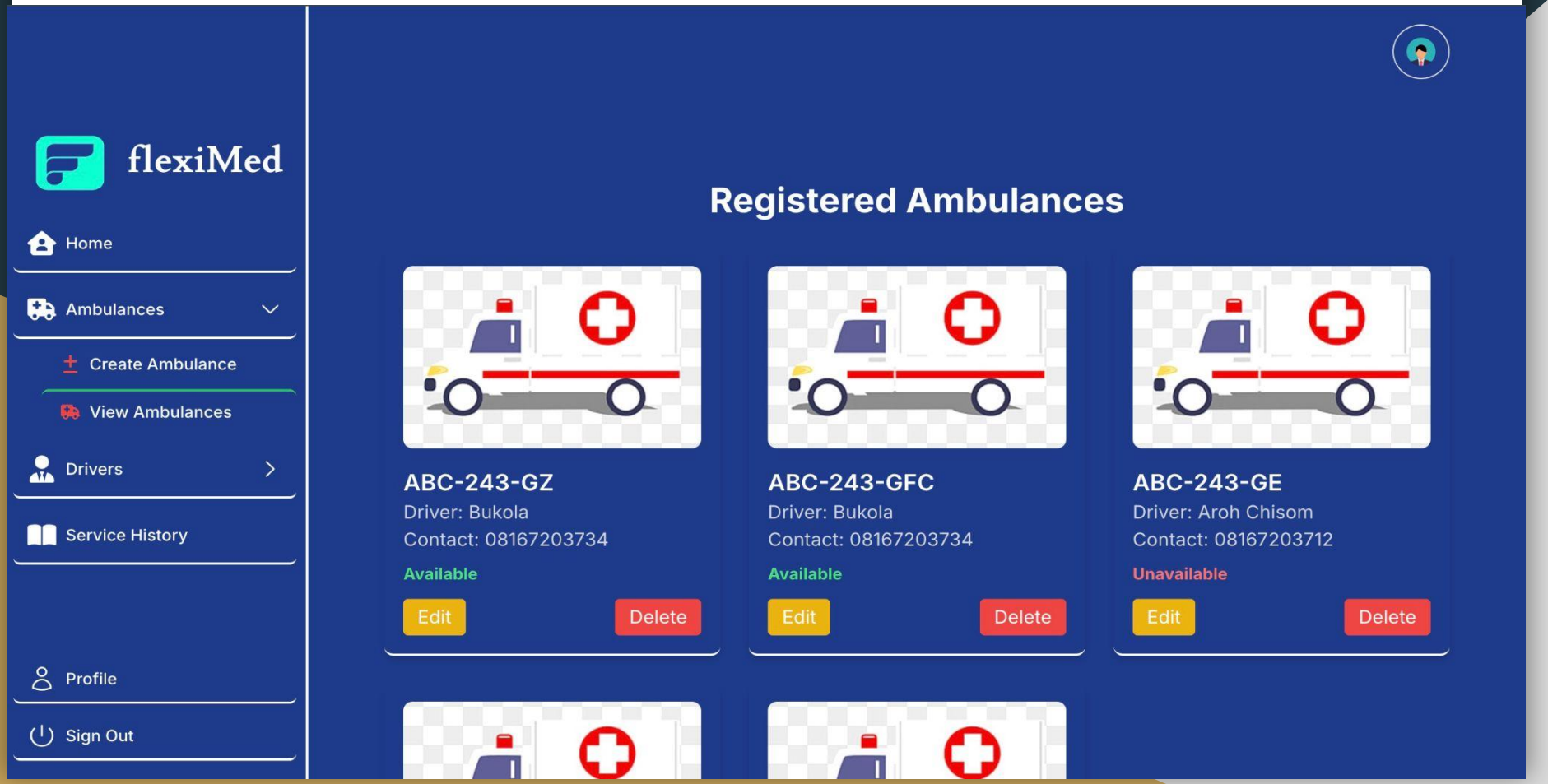
A user with role(Dispatcher) can register an ambulance and assign the ambulance to a driver



Live Demonstration

All Ambulances page

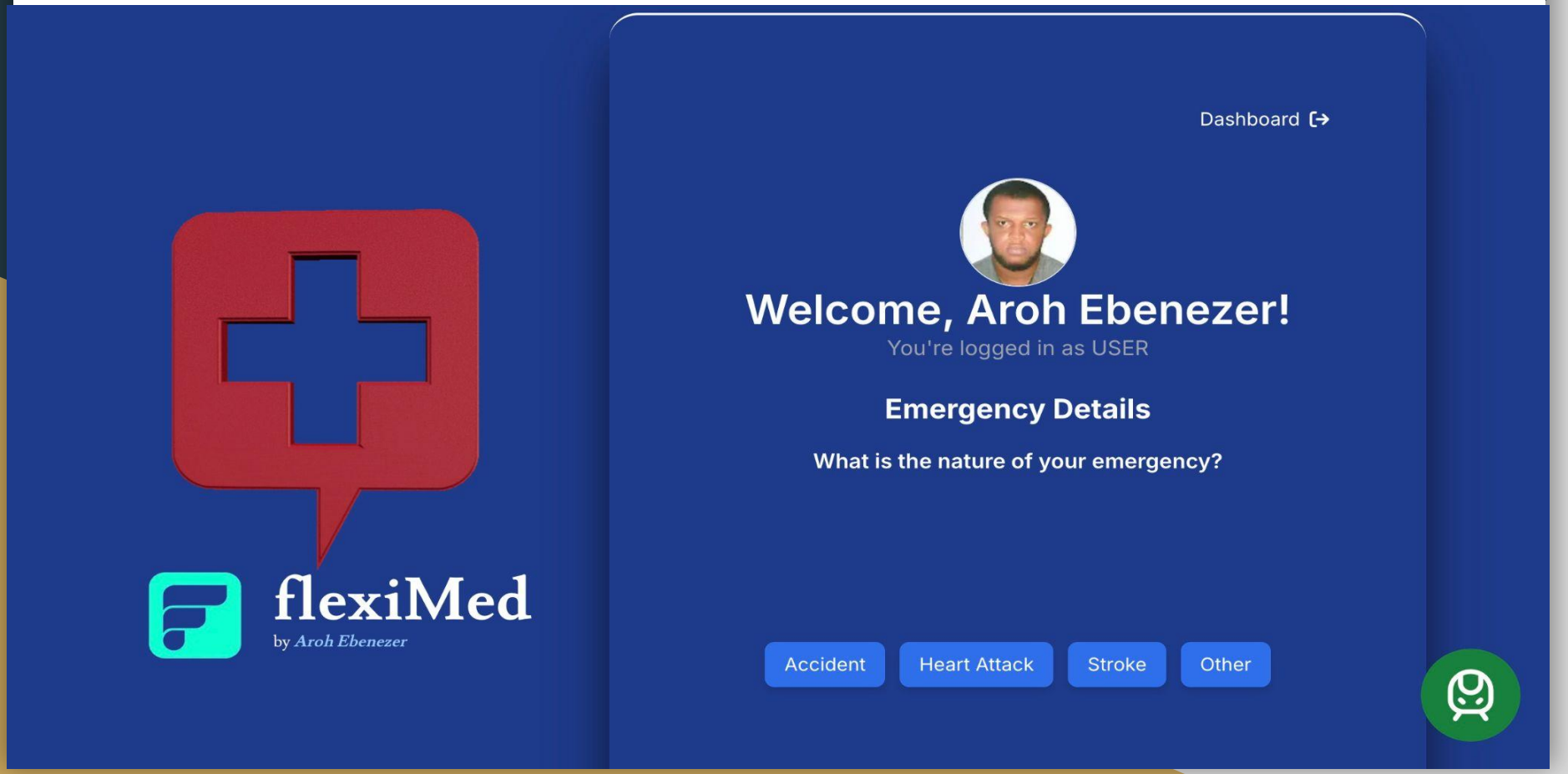
A user with role(Dispatcher) can see all registered ambulances and update any



Live Demonstration

Users Home page

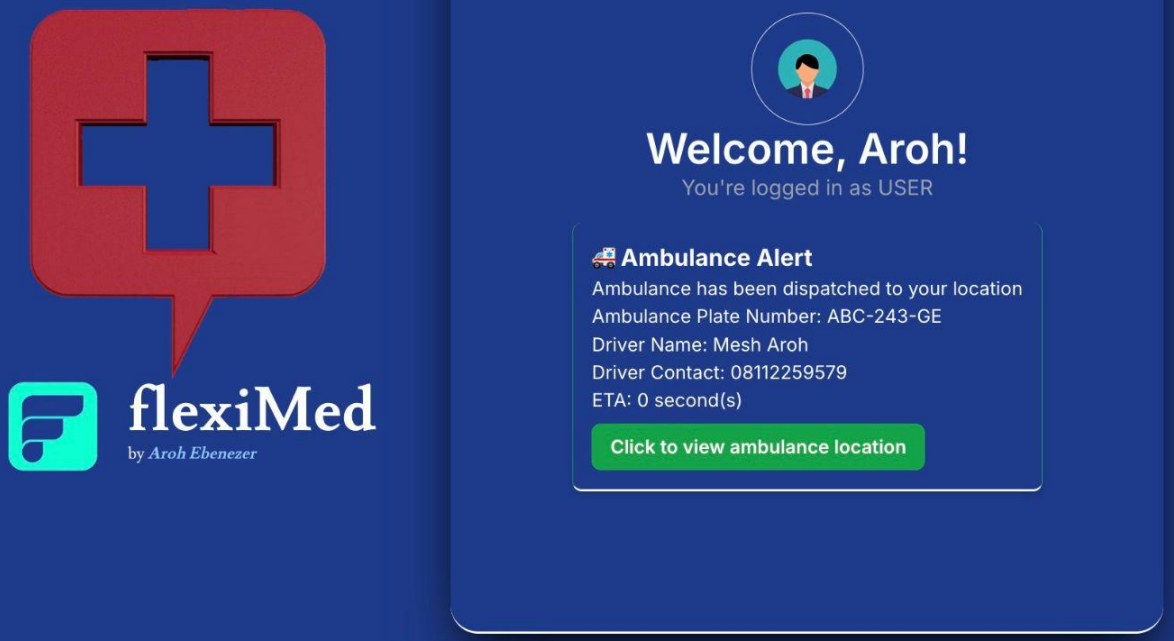
- A user can sign in with google auth or use basic registration and login
- User answers emergency and medical history related questions
- User submits the request



Live Demonstration

Users Notification page

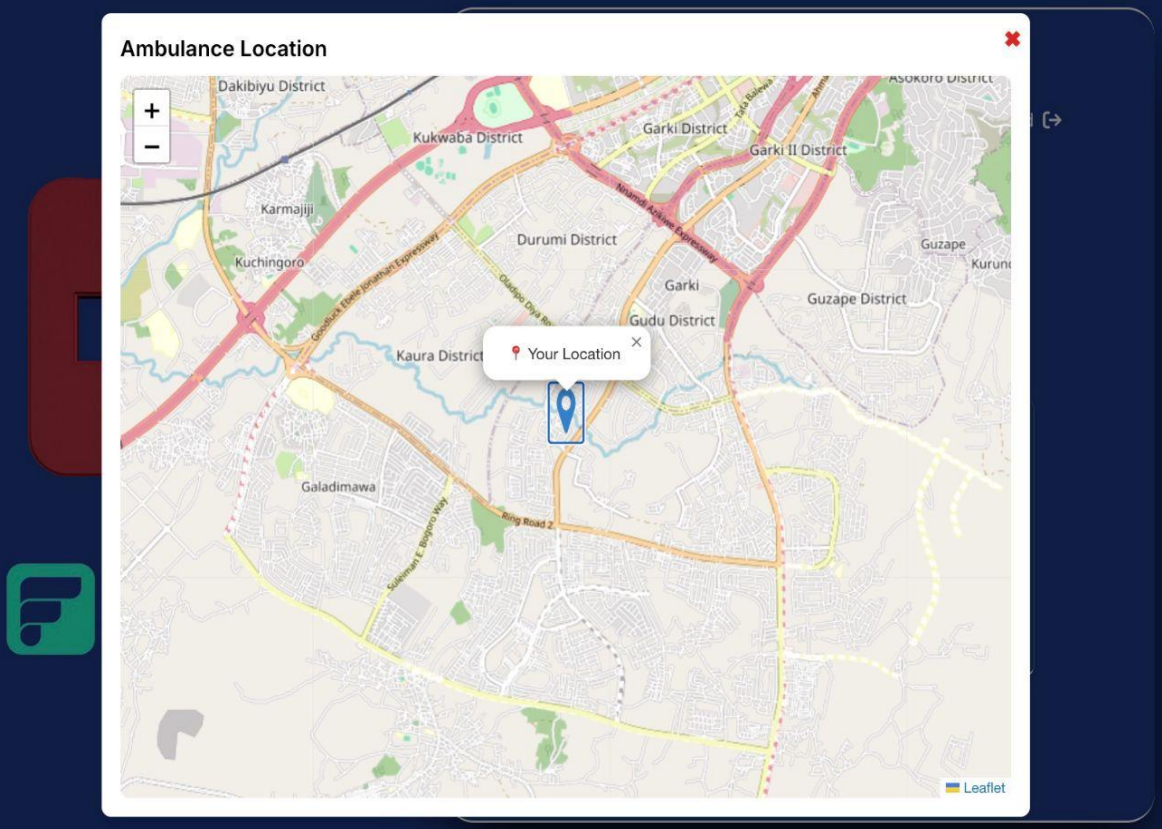
- User receives ambulance dispatch notification



Live Demonstration

Users Ambulance tracking leaflet

- User can track ambulance location in real time



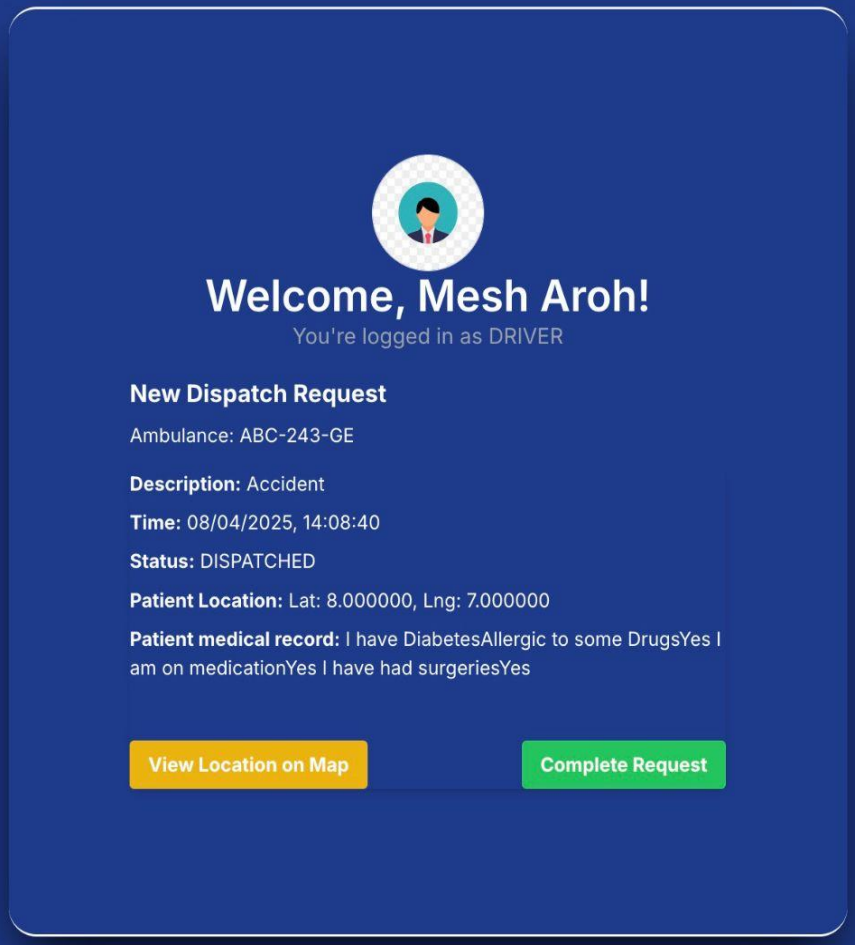
Live Demonstration

Drivers Home Page

- Ambulance driver receives dispatch request

flexiMed

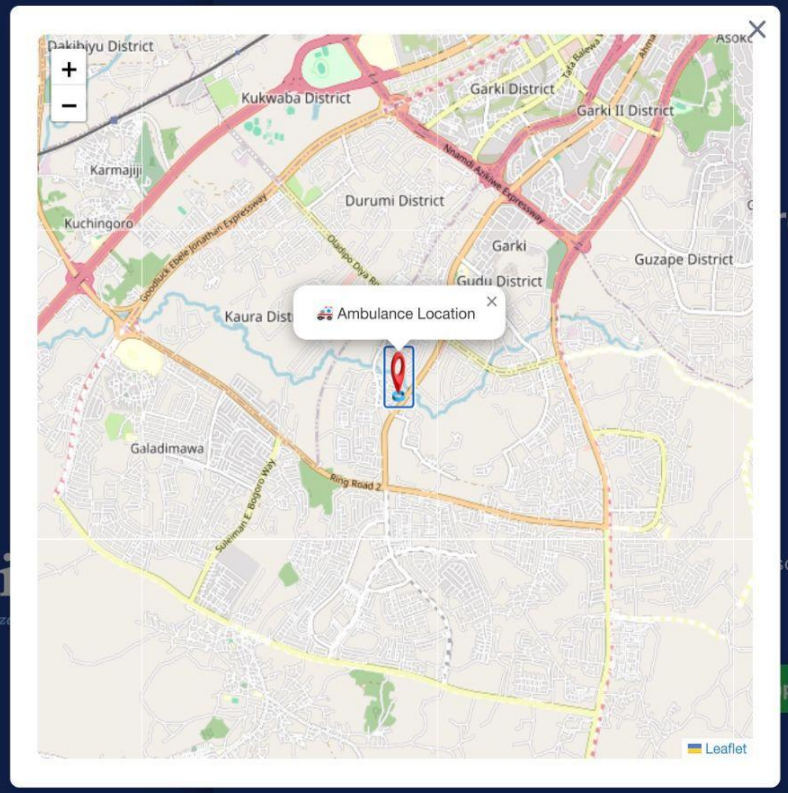
by *Aroh Ebenezer*



Live Demonstration

User Location Tracking Leaflet

- Ambulance driver can locate user on leaflet



Live Demonstration

Drivers Home Page With no Dispatch Request

- Ambulance driver can complete request and refresh to fetch new requests

flexiMed
by *Aroh Ebenezer*